Final Project

- Peiyu Zhong, Kelsey Zeng, Zihan Ni

Maximal Matching Algorithm <O(E + V)>

1. `maximalMatching(graph: Graph[Int, Int]): List[(Long, Long)]`

   - Purpose: Finds a maximal matching in a graph.
   - Method: Collects all edges, filters by unmatched vertex pairs, and maintains a set of matched vertices.
   - Output: List of edges that form the maximal matching.

2. `saveMatching(matching: List[(Long, Long)], outputFile: String)`

   - Purpose: Saves the matching to a CSV file.
   - Method: Converts matching list to DataFrame, then writes to file using DataFrame operations.
   - Output: CSV file containing the matching pairs.

3. `verifyMatching(graph: Graph[Int, Int]): Boolean`

   - Purpose: Verifies if the matching is independent and maximal.
   - Method: Uses flatMap to check for independent edges and join operations to ensure all unmatched vertices are adjacent to matched vertices.
   - Output: Boolean indicating if the matching meets criteria.

Greedy Matching Algorithm <O(ElogE)>

- Initialization:
  - `val matchedVertices = mutable.HashSet[VertexId]()`
    - The algorithm starts by initializing a data structure to keep track of which vertices have been matched already. This is crucial to ensure no vertex is included in more than one matching edge.
  - `val sortedEdges = graphInput.edges.map(e => (e, Random.nextDouble())).sortBy(_._2).map(_._1).cache()`
  - The input graph is preprocessed to sort its edges in a randomized order. This preprocessing ensures each execution of the algorithm can potentially yield different results and avoids any inherent bias in edge ordering.

Greedy continued

- Processing the Edges:
  - The algorithm processes each edge from the sorted list one by one.
  - For each edge:
    - The source and destination vertices of the edge are checked to see if they have been matched.
    - If both vertices are unmatched, this edge is eligible to be part of the matching:
    - The edge is added to the final matching result.
    - Both vertices are marked as matched, so they cannot be part of any subsequent edges.

Result

	log_normal_100.csv	musae_ENGB_edges.	soc-pokec-relationships.csv	soc-LiveJournal1.csv	twitter_original_edges.csv	com-orkut.ungraph.csv
MaximalMatching Reported	48	2443	588542	1502622	91747	1321378
MaximalMatching Verified	48	2443	588542	1502622	91747	1321378
MaximalMatching Time Consu	0.2s in local machine	0.38s in local machine	65s in 2*4 N1 Core	229s in 2*4 N1 Core	23s in local machine (16GB)	48s in local machine (20GB)
Greedy Reported	49	2249	588250	1546075	91797	1325470
Greedy Verified	49	2249	588250	1546075	91797	1325470
Greedy Time Consumption	0.18s in local machine	0.28s in local machine	4s in local machine	8s in local machine	8s in local machine	31s in local machine

Advantages for Greedy Approach

Simplicity:

The greedy matching algorithm is straightforward and easy to understand. It iteratively selects edges for matching by checking if neither vertex is already matched, making it conceptually simple.

Efficiency:

The algorithm works efficiently by processing each edge only once. The use of a HashSet for tracking matched vertices ensures constant-time checks, allowing for rapid selection.

Randomization:

The inclusion of random shuffling within each partition helps reduce bias in edge selection, ensuring a fair distribution of matches across partitions.

Parallel Processing:

The algorithm is designed to work seamlessly with distributed computing frameworks like Apache Spark, leveraging partitions and caching to handle large datasets efficiently.

Advantages for MM

Comprehensive:

The maximal matching algorithm aims to produce a valid and maximal matching by ensuring that all selected edges form a subset of the input graph and that no vertex has a degree greater than 1.

Direct Approach:

The algorithm directly iterates over edges and vertices, filtering them to produce a matching. This eliminates the need for random shuffling or additional processing steps, making it more streamlined.

Explicit Checks:

The algorithm includes explicit checks for independent and maximal properties, ensuring that each vertex is part of the matching or has a neighbor that is. This provides strong guarantees for the output.

Comparison and Conclusion:

Greedy Matching:

Offers simplicity, efficiency, and adaptability, making it suitable for general graph matching tasks. However, its time complexity can increase significantly with the logarithmic factor.

Maximal Matching:

Provides comprehensive and balanced processing, ensuring valid and maximal matchings while maintaining linear time complexity, particularly suitable for balanced or edge-dominated graphs.

Use Cases:

Greedy Matching: Works well for general, distributed tasks or when simplicity is prioritized.

Maximal Matching: Suited for applications requiring valid and maximal matchings with explicit checks, such as scheduling or optimization.

New Data...

1. General Maximal Matching/Greedy Matching:

   For balanced datasets, a general maximal matching algorithm is preferred. This ensures accurate coverage, allowing for more comprehensive results without compromising integrity.

   This approach ensures that each matching solution is valid and maximal, meaning no further edges can be added without violating the properties of a matching.

2. Luby's Algorithm:

   For larger datasets, particularly those in the realm of big data, Luby's algorithm can be used. This allows for more efficient processing by compromising some accuracy to provide a raw estimate.

   Efficient Convergence:

   The algorithm converges quickly because it iteratively reduces the set of active vertices, making the graph sparser with each iteration. This leads to fewer vertices to process in subsequent iterations, accelerating convergence.

TO-DO

Adaptation Process:

The existing Luby's MIS algorithm can be adapted to compute a maximal matching by focusing on edge selections rather than vertex sets.

This adapted algorithm can be used effectively for big data scenarios, providing a balance between efficiency and functionality.